시스템 프로그래밍

담당 교수: 김지환

< Project 3 >

SIC/XE Machine Linking Loader 구현

서강대학교 컴퓨터공학과

학번: 20161577

이름: 김인호

제출 마감:  
2018년 04월 30일 (월)

**목차**

[1. 프로그램 개요 5](#_Toc513063348)

[2. 프로그램 설명 5](#_Toc513063349)

[2.1. 프로그램 흐름도 5](#_Toc513063350)

[2.2. Linking Loader 알고리즘 설명 6](#_Toc513063351)

[2.2.1. Pass1 – External Symbol Table 생성 6](#_Toc513063352)

[2.2.2. Pass2 – linking 및 loading 작업 6](#_Toc513063353)

[3. 모듈 정의 7](#_Toc513063354)

[3.1. 모듈 이름: pAddrCMD(INPUT\_CMD) 7](#_Toc513063355)

[3.1.1. 기능 7](#_Toc513063356)

[3.1.2. 사용 변수 7](#_Toc513063357)

[3.2. 모듈 이름: loaderCMD(INPUT\_CMD) 7](#_Toc513063358)

[3.2.1. 기능 7](#_Toc513063359)

[3.2.2. 사용 변수 7](#_Toc513063360)

[3.3. 모듈 이름: linkLoaderPass1(FILE\*\*) 7](#_Toc513063361)

[3.3.1. 기능 7](#_Toc513063362)

[3.3.2. 사용 변수 7](#_Toc513063363)

[3.4. 모듈 이름: linkLoaderPass2(FILE\*\*) 8](#_Toc513063364)

[3.4.1. 기능 8](#_Toc513063365)

[3.4.2. 사용 변수 8](#_Toc513063366)

[3.5. 모듈 이름: fcloseObj(FILE\*\*) 8](#_Toc513063367)

[3.5.1. 기능 8](#_Toc513063368)

[3.5.2. 사용 변수 9](#_Toc513063369)

[3.6. 모듈 이름: searchCS(char\*) 9](#_Toc513063370)

[3.6.1. 기능 9](#_Toc513063371)

[3.6.2. 사용 변수 9](#_Toc513063372)

[3.7. 모듈 이름: searchES(char\*) 9](#_Toc513063373)

[3.7.1. 기능 9](#_Toc513063374)

[3.7.2. 사용 변수 9](#_Toc513063375)

[3.8. 모듈 이름: printCntSecTable(void\*) 10](#_Toc513063376)

[3.8.1. 기능 10](#_Toc513063377)

[3.8.2. 사용 변수 10](#_Toc513063378)

[3.9. 모듈 이름: printExtSym(void\*) 10](#_Toc513063379)

[3.9.1. 기능 10](#_Toc513063380)

[3.9.2. 사용 변수 10](#_Toc513063381)

[3.10. 모듈 이름: bpCMD(INPUT\_CMD) 10](#_Toc513063382)

[3.10.1. 기능 10](#_Toc513063383)

[3.10.2. 사용 변수 10](#_Toc513063384)

[3.11. 모듈 이름: searchBP(int) 11](#_Toc513063385)

[3.11.1. 기능 11](#_Toc513063386)

[3.11.2. 사용 변수 11](#_Toc513063387)

[3.12. 모듈 이름: runCMD() 11](#_Toc513063388)

[3.12.1. 기능 11](#_Toc513063389)

[3.12.2. 사용 변수 11](#_Toc513063390)

[3.13. 모듈 이름: dumpReg() 11](#_Toc513063391)

[3.13.1. 기능 11](#_Toc513063392)

[3.13.2. 사용 변수 11](#_Toc513063393)

[3.14. 모듈 이름: getTargetAddress(int, FMT) 12](#_Toc513063394)

[3.14.1. 기능 12](#_Toc513063395)

[3.14.2. 사용 변수 12](#_Toc513063396)

[3.15. 모듈 이름: SICAddress(int) 12](#_Toc513063397)

[3.15.1. 기능 12](#_Toc513063398)

[3.15.2. 사용 변수 12](#_Toc513063399)

[3.16. 모듈 이름: immediateAddress(int, FMT) 12](#_Toc513063400)

[3.16.1. 기능 12](#_Toc513063401)

[3.16.2. 사용 변수 12](#_Toc513063402)

[3.17. 모듈 이름: indirectAddress(int, FMT) 13](#_Toc513063403)

[3.17.1. 기능 13](#_Toc513063404)

[3.17.2. 사용 변수 13](#_Toc513063405)

[3.18. 모듈 이름: simpleAddress(int, FMT) 13](#_Toc513063406)

[3.18.1. 기능 13](#_Toc513063407)

[3.18.2. 사용 변수 13](#_Toc513063408)

[3.19. 모듈 이름: getMem(int, int) 13](#_Toc513063409)

[3.19.1. 기능 13](#_Toc513063410)

[3.19.2. 사용 변수 13](#_Toc513063411)

[3.20. 모듈 이름: putMem(int, int, int) 14](#_Toc513063412)

[3.20.1. 기능 14](#_Toc513063413)

[3.20.2. 사용 변수 14](#_Toc513063414)

[4. 전역 변수 정의 14](#_Toc513063415)

[4.1. int progAddr 14](#_Toc513063416)

[4.2. LIST extSymTable 14](#_Toc513063417)

[4.3. LIST breakPntList 14](#_Toc513063418)

[4.4. int endAddress 14](#_Toc513063419)

[4.5. int registers[REG\_CNT] 15](#_Toc513063420)

[5. 코드 15](#_Toc513063421)

[5.1. 20161577.h 15](#_Toc513063422)

[5.2. 20161577.c 18](#_Toc513063423)

[5.3. assembler.h 21](#_Toc513063424)

[5.4. assembler.c 22](#_Toc513063425)

[5.5. cmdProc.h 38](#_Toc513063426)

[5.6. cmdProc.c 39](#_Toc513063427)

[5.7. execute.h 43](#_Toc513063428)

[5.8. execute.c 45](#_Toc513063429)

[5.9. hash.h 54](#_Toc513063430)

[5.10. hash.c 55](#_Toc513063431)

[5.11. linkedList.h 57](#_Toc513063432)

[5.12. linkedList.c 58](#_Toc513063433)

[5.13. linkLoader.h 60](#_Toc513063434)

[5.14. linkLoader.c 60](#_Toc513063435)

[5.15. memory.h 66](#_Toc513063436)

[5.16. memory.c 66](#_Toc513063437)

[5.17. shell.h 68](#_Toc513063438)

[5.18. shell.c 68](#_Toc513063439)

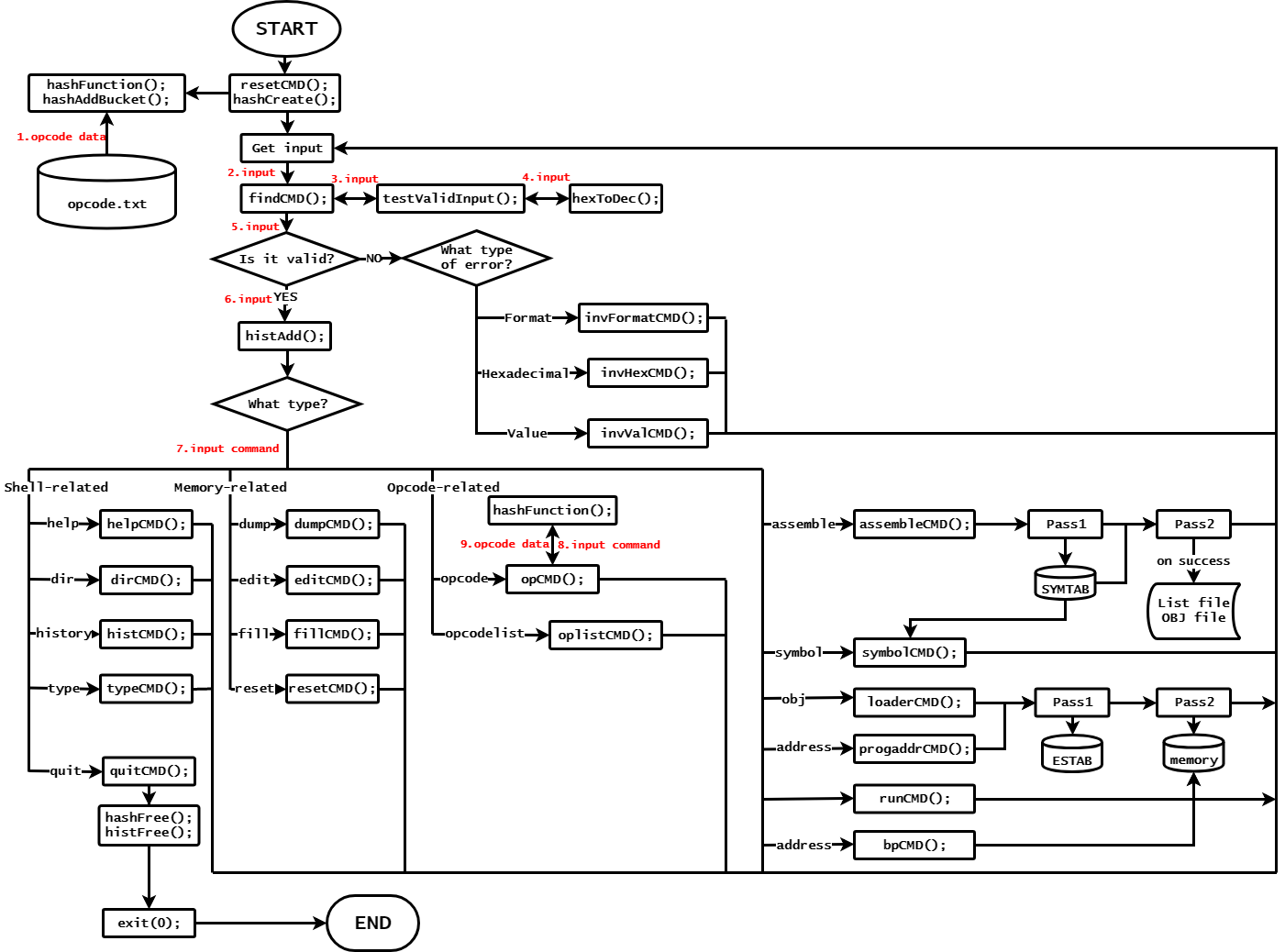
[5.19. Makefile 70](#_Toc513063440)

# 프로그램 개요

사전에 구현한 shell 환경과 assembler에 추가되는 기능으로, SIC/XE Machine Linking Loader를 구현한다. Control section 단위로 나눠진 여러 object file을 linking하여 메모리에 load하는 작업과 loading된 프로그램을 실행하고 breakpoint을 통한 디버깅 기능까지 구현한다.

# 프로그램 설명

## 프로그램 흐름도



## Linking Loader 알고리즘 설명

구현된 SIC/XE Machine Assembler는 주어진 .asm 소스파일을 총 2번의 읽는 과정 (Pass1과 Pass2)을 거쳐 list 파일과 object 파일을 생성하는 알고리즘이다.

### Pass1 – External Symbol Table 생성

한 프로그램의 각 control section이 assemble 된 object file이 주어진 것으로 시작한다. 우선 정해진 프로그램의 시작 주소를 첫 control section의 시작 주소로 놓는다. 각 control section에 대한 object file을 읽으면서 header record로부터 해당 control section의 길이를 알아내고 시작 주소(CSADDR)에 이 길이(CSLTH)를 더한 값이 다음 control section의 시작 주소가 된다.

또한 각 control section object file의 D record를 읽으면서 해당 control section에 정의된 각 변수 및 상수의 offset을 CSADDR에 더하여 각각의 주소를 얻는다. 각 control section의 시작 주소와 각 변수 및 상수는 external symbol table (ESTAB)에 저장된다. 추가적으로 각 control section의 이름과 각 변수 및 상수에 대한 중복이 없는지 확인하면서 에러 처리를 한다.

### Pass2 – linking 및 loading 작업

CSADDR를 계산하고 다루는 방법은 pass1과 동일하다. 먼저 R record에 reference number 형태로 나오는 external references에 대하여 ESTAB에 탐색하여 주소를 얻는다. 발견이 안 되면 에러이다.

그 후 T record를 각각이 해당되는 offset 주소 메모리에 올린다. 마지막으로 M record를 읽으면서 각 T record의 필요한 modification을 진행한다.

# 모듈 정의

## 모듈 이름: pAddrCMD(INPUT\_CMD)

### 기능

사용자가 입력한 주소를 프로그램의 시작 주소로 설정한다.

### 사용 변수

|  |  |
| --- | --- |
| INPUT\_CMD ipcmd | 함수 인자, 사용자의 입력을 저장한 변수 |

## 모듈 이름: loaderCMD(INPUT\_CMD)

### 기능

설정된 시작 주소로부터 주어진 obj 파일에 대한 linking loader 작업을 수행한다.

### 사용 변수

|  |  |
| --- | --- |
| INPUT\_CMT ipcmd | 함수 인자, 사용자 입력 저장 |
| FILE\* objFptr[] | object file pointer |
| int progLen | 프로그램의 총 길이 |

## 모듈 이름: linkLoaderPass1(FILE\*\*)

### 기능

앞서 설명된 linking loader의 Pass1 알고리즘을 수행하는 함수이다. ESTAB을 생성한다.

### 사용 변수

|  |  |
| --- | --- |
| FILE\*\* objFptr | 함수 인자, object file pointer |
| int CSADDR | control section 시작 주소 |
| char record[] | 읽어온 record 저장 |
| char csName[] | control section이름 |
| char esName[] | external symbol 이름 |
| char addr[] | 각 symbol의 주소 저장 |
| CNT\_SEC\* newCntSec | 새 control section node pointer |
| EXT\_SYMBOL\* newExtSym | 새 external symbol node pointer |

## 모듈 이름: linkLoaderPass2(FILE\*\*)

### 기능

앞서 설명된 linking loader의 Pass2 알고리즘을 수행하는 함수이다. ESTAB의 정보로 linking loader 작업을 수행한다.

### 사용 변수

|  |  |
| --- | --- |
| FILE\*\* objFptr | 함수 인자, object file pointer |
| int CSADDR | control section 시작 주소 |
| int EXECADDR | 프로그램 시작 주소 |
| int CSLTH | control section 길이 |
| int offset | 시작 주소로부터의 거리 |
| int tLen | T record의 길이 |
| int hByteCnt | Modification할 half byte의 개수 |
| int modAddress | Modification할 주소 |
| int maxExSymIndex | 최대 reference number |
| int\* refNum | reference number와 각각의 주소 저장 array |
| char record[] | 읽어온 record |
| char esName[] | external symbol 이름 |
| char temp[] | record parsing을 위한 임시 저장소 |
| NODE\* curCntSec | 현재 control section pointer |

## 모듈 이름: fcloseObj(FILE\*\*)

### 기능

Linking loader 작업에 사용된 object 파일을 모두 닫아주는 함수이다.

### 사용 변수

|  |  |
| --- | --- |
| FILE\*\* objFptr | 함수 인자, 파일 포인터 배열 |

## 모듈 이름: searchCS(char\*)

### 기능

ESTAB애서 control section 이름을 찾는다. 찾으면 true 없으면 false를 return한다.

### 사용 변수

|  |  |
| --- | --- |
| char\* csName | 함수 인자, ESTAB에서 찾을 control section 이름 |
| NODE\* cur | ESTAB 탐색용 pointer |
| CNT\_SEC\* data | 각 node의 data |

## 모듈 이름: searchES(char\*)

### 기능

ESTAB에서 external symbol 이름을 찾는다. 찾으면 그의 주소를 없으면 -1을 return 한다.

### 사용 변수

|  |  |
| --- | --- |
| char\* esName | 함수 인자, ESTAB에서 찾을 external symbol 이름 |
| NODE\* curCS | ESTAB 탐색용 pointer |
| NODE\* curES | ESTAB 탐색용 pointer |

## 모듈 이름: printCntSecTable(void\*)

### 기능

Linking loader의 작업이 성공적으로 완료 했을 때 ESTAB의 control section에 대한 내용을 형식에 맞추어 출력한다.

### 사용 변수

|  |  |
| --- | --- |
| void\* data | 함수 인자, control section 정보를 갖는 포인터 |

## 모듈 이름: printExtSym(void\*)

### 기능

Linking loader의 작업이 성공적으로 완료 했을 때 ESTAB의 external symbol에 대한 내용을 형식에 맞추어 출력한다.

### 사용 변수

|  |  |
| --- | --- |
| void\* data | 함수 인자, external symbol 정보를 갖는 포인터 |

## 모듈 이름: bpCMD(INPUT\_CMD)

### 기능

bp 명령어 입력 시 호출되는 함수로 breakpoint를 생성하거나 지울 수 있고 또한 생성된 breakpoint의 목록도 볼 수 있다.

### 사용 변수

|  |  |
| --- | --- |
| INPUT\_CMD ipcmd | 함수 인자, 사용자 입력 저장 |
| BREAK\_PNT\* newBP | 새로 추가 될 break point의 정보 포인터 |

## 모듈 이름: searchBP(int)

### 기능

Break point linked list를 탐색하여 해당 주소가 있을 경우 true 아니면 false를 return 한다.

### 사용 변수

|  |  |
| --- | --- |
| int address | 함수 인자, 찾을 주소 |
| NODE\* curBP | linked list 탐색용 pointer |

## 모듈 이름: runCMD()

### 기능

run 명령어 입력 시 수행되는 함수로 지정된 시작주소부터 메모리를 읽으며 프로그램을 수행한다.

### 사용 변수

|  |  |
| --- | --- |
| static curAddress | 현재 수행 중인 지점의 메모리 주소 |
| static lastBP | 마지막 breakpoint 주소 기억 |
| int targetVal | instruction에서 해석된 target value |
| int targetAddress | instruction에서 해석된 target address |
| OBJ curObj | instruction을 해석한 정보를 저장 |

## 모듈 이름: dumpReg()

### 기능

Register에 저장된 값을 화면에 출력해주는 함수이다

### 사용 변수

사용한 변수 없음.

## 모듈 이름: getTargetAddress(int, FMT)

### 기능

instruction의 format과 주소를 바탕으로 target address를 구하는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int curAddress | 함수 인자, instruction의 주소 |
| FMT format | 함수 인자, instruction의 format |
| ADR\_MODE addrMode | instruction 의 addressing mode |
| int target | target address를 저장하는 변수 |

## 모듈 이름: SICAddress(int)

### 기능

SIC instruction일 경우에 대해 target address를 구해주는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int curAddress | 함수 인자, instruction의 주소 |

## 모듈 이름: immediateAddress(int, FMT)

### 기능

Immediate addressing일 경우에 대해 target address를 구해주는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int curAddress | 함수 인자, instruction의 주소 |
| FMT format | 함수 인자, instruction의 format |

## 모듈 이름: indirectAddress(int, FMT)

### 기능

Indirect addressing일 경우에 대해 target address를 구해주는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int curAddress | 함수 인자, instruction의 주소 |
| FMT format | 함수 인자, instruction의 format |
| int target | target address 저장 변수 |

## 모듈 이름: simpleAddress(int, FMT)

### 기능

Simple addressing일 경우에 대해 target address를 구해주는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int curAddress | 함수 인자, instruction의 주소 |
| FMT format | 함수 인자, instruction의 format |
| int target | target address 저장 변수 |
| int setBit | b와 p bit의 mask를 씌운 결과 |

## 모듈 이름: getMem(int, int)

### 기능

메모리의 주어진 주소로부터 일정 half byte만큼 공간에 있는 값을 반환해주는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int address | 함수 인자, 가져올 메모리 주소 |
| int hBytes | 함수 인자, 가져올 half byte 개수 |
| int val | 값을 누적하여 return해줄 값 |

## 모듈 이름: putMem(int, int, int)

### 기능

메모리의 해당 주소에 일정 byte만큼 저장해주는 함수.

### 사용 변수

|  |  |
| --- | --- |
| int address | 함수 인자, 값이 저장될 목표 주소 |
| int bytes | 함수 인자, 값이 메모리에서 차지할 byte 개수 |
| int value | 함수 인자, 저장할 값 |

# 전역 변수 정의

## int progAddr

프로그램의 시작 주소를 저장하는 변수. Default 값은 0x0이다.

## LIST extSymTable

External symbol table을 다차원 linked list 형태로 저장한 자료구조의 head pointer.

## LIST breakPntList

Breakpoint의 목록을 저장하는 linked list의 head pointer.

## int endAddress

프로그램의 끝 주소를 저장하는 변수.

## int registers[REG\_CNT]

A, X, L, B, S, T, F, PC, SW 차례로 총 9개의 register의 값을 저장하는 배열.

# 코드

## 20161577.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: 20161577.h \*

\* File description: Main header file for the project. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include <stdio.h>

#include <string.h>

#include <ctype.h>

#include <stdlib.h>

#include <math.h>

#include <dirent.h>

#include <sys/types.h>

#include <sys/stat.h>

#define CMD\_LEN 257 // maximum length of input string

#define CMD\_CNT 18 // list of command formats

#define ARG\_MAX 3 // maximum argument count

#define MEM\_SIZE 1048576 // 2^20 = 1MB

#define HASH\_SIZE 20 // hash table size

#define CS\_LEN 7 // maximum control section name length

#define CS\_MAX 3 // maximum control section count

#define ASM\_LEN 121 // maximum length of single asm source line

#define INC\_BYTE 16 \* 16 // increase byte in object code

#define INC\_HBYTE 16 // increase half byte in object code

#define MAX12DISP 4096 // highest number expressable with 12 bits

#define MAX15ADDR 32768 // highest number expressable with 15 bits

#define REG\_CNT 9 // register count

#define Areg 0 // A register number

#define Xreg 1 // X register number

#define Lreg 2 // L register number

#define Breg 3 // B register number

#define Sreg 4 // S register number

#define Treg 5 // T register number

#define Freg 6 // F register number

#define PCreg 7 // PC register number

#define SWreg 8 // SW register number

typedef struct dirent ENTRY; // for dir command

typedef struct stat STBUF; // for dir command

typedef enum { false, true } bool; // workaround for bool type in C

// error codes, for better readability

typedef enum {

SAFE, FORMAT, HEX, VALUE, FILENAME

} ERROR\_CODE;

// command types, for better readability

typedef enum {

shell, memory, opcode, assembler, linkLoader, invalid

} CMD\_TYPE;

// command functions, for better readablitiy

typedef enum {

help, dir, quit, hist, dump, edit, fill, reset, op, oplist, type, assemble, symbol, pAddr, loader, run, bp, invFormat, invHex, invVal, invFile

} CMD\_FUNC;

// assembly source error type, for better readability

typedef enum {

OK, SYMBOL, INSTRUCTION, OPERAND

} ASM\_ERROR;

// directives list, for better readability

typedef enum {

NOTDR, START, END, BASE, BYTE, WORD, RESB, RESW

} DIREC\_NAME;

// register list, for better readability

typedef enum {

A, X, L, B, S, T, F, PC, SW

} REG;

// command format structure

typedef struct {

char str[CMD\_LEN];

char abb[CMD\_LEN];

CMD\_TYPE category;

CMD\_FUNC func;

bool arg;

} COMMAND;

// user input command parsed structure

typedef struct {

CMD\_FUNC cmd;

short argCnt;

char arg[ARG\_MAX][10];

} INPUT\_CMD;

// structure for generic linked list

typedef struct NODE {

void\* data;

struct NODE\* next;

} NODE;

typedef NODE\* LIST;

// history node structure

typedef struct HIST\_STRUCT {

char str[CMD\_LEN];

} HIST\_NODE;

// hash table bucket structure

typedef struct HASH\_STRUCT {

int codeVal;

char code[3];

char inst[CMD\_LEN];

enum { f1, f2, f34 } format;

int operandCnt;

struct HASH\_STRUCT\* next;

} HASH\_ENTRY;

// assembly source parse structure

typedef struct ASM\_STRUCT {

char source[ASM\_LEN]; // source code

char label[ASM\_LEN]; // label field

char inst[ASM\_LEN]; // instruction field

char operand[2][ASM\_LEN]; // operand field

bool hasLabel; // flag for having label

bool indexing; // flag for having indexing mode

int lineNum; // line number in list file

int location; // location counter of instruction

int operandCnt; // operand count

int byteSize; // size of instruction in bytes

unsigned objCode; // value of object code

enum { ERROR, INST, PSEUDO, COMMENT } type; // type of source line

enum { NONE, format1, format2, format3, format4 } format; // type of instruction

DIREC\_NAME direcName; // directive type

ASM\_ERROR errorCode; // error type

struct ASM\_STRUCT\* next; // pointer to next assembly source

} ASM\_SRC;

// symbol table entry structure

typedef struct SYMBOL\_STRUCT {

char symbol[ASM\_LEN];

int address;

struct SYMBOL\_STRUCT\* next;

} SYMBOL\_ENTRY;

// Object Code list structure

typedef struct OBJ\_STRUCT {

unsigned objCode;

int location;

int byteSize;

struct OBJ\_STRUCT\* next;

} OBJ\_CODE;

// Modification Record list structure

typedef struct MOD\_STRUCT {

int location;

int lenHB;

struct MOD\_STRUCT\* next;

} MOD\_RECORD;

// Control Section structure for ESTAB

typedef struct CS\_STRUCT {

char csName[CS\_LEN];

int stAddress;

int length;

LIST extSym;

} CNT\_SEC;

// External Symbol structure for ESTAB

typedef struct ES\_STRUCT {

char symName[CS\_LEN];

int address;

} EXT\_SYMBOL;

// Break Point list structure

typedef struct BP\_STRUCT {

int address;

} BREAK\_PNT;

int progAddr; // program start address set by progaddr command

int execAddress, endAddress; // execution and end address

int registers[REG\_CNT]; // A, X, L, B, S, T, F, PC, SW

int hexToDec(char\*); // function to check for vaild hex value and return converted decimal value

void symTableFree(); // function to free SYMTAB

void parseListFree(); // function to free ASM parse list

void objListFree(); // function to free OBJ code list

void modListFree(); // function to free Modification Record list

HASH\_ENTRY\* bucketSearch(char\*); // function to search bucket

## 20161577.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: 20161577.c \*

\* File description: Main file for the project. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "cmdProc.h"

#include "shell.h"

#include "memory.h"

#include "hash.h"

#include "assembler.h"

#include "linkLoader.h"

#include "execute.h"

int main() {

char inp[CMD\_LEN]; // input string

char tmp[CMD\_LEN]; // temporary string to copy input

int i, j;

INPUT\_CMD input; // storage for parsed input

for(i = 0; i < REG\_CNT; i++)

registers[i] = 0; // initialize registers

resetCMD(); // initialize memory

hashCreate(); // create hash table of opcodes

progAddr = 0; // default program starting address

while(1) {

printf("sicsim> ");

fgets(inp, CMD\_LEN, stdin); // get input string

inp[strlen(inp) - 1] = '\0'; // replace \n with null character

// copy input string to tmp but place one space before and after comma ','

j = 0;

for(i = 0; inp[i]; i++) {

if(inp[i] == ',') {

strcpy(tmp + j, " , "); // place space around commma ','

j += 3;

}

else

tmp[j++] = inp[i];

}

tmp[j] = '\0';

input = findCMD(tmp); // find the command format from input string

switch(input.cmd) {

case assemble:

case loader:

case bp:

case invFormat:

case invHex:

case invVal:

case invFile:

break;

default:

histAdd(inp); // if command is not invalid add to history

}

// call function for each command

switch(input.cmd) {

case help:

helpCMD();

break;

case dir:

dirCMD();

break;

case quit:

quitCMD();

break;

case hist:

histCMD();

break;

case dump:

dumpCMD(input);

break;

case edit:

editCMD(input);

break;

case fill:

fillCMD(input);

break;

case reset:

resetCMD();

break;

case op:

opCMD(input);

break;

case oplist:

oplistCMD();

break;

case type:

typeCMD(input);

break;

case assemble:

if(assembleCMD(input))

histAdd(inp);

break;

case symbol:

symbolCMD();

break;

case pAddr:

pAddrCMD(input);

break;

case loader:

if(loaderCMD(input))

histAdd(inp);

break;

case run:

runCMD();

break;

case bp:

if(bpCMD(input))

histAdd(inp);

break;

case invFormat:

invFormatCMD();

break;

case invHex:

invHexCMD();

break;

case invVal:

invValCMD();

break;

case invFile:

invFileCMD();

break;

}

}

return 0;

}

## assembler.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: assembler.h \*

\* File description: Header file for assembler tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

bool assembleCMD(INPUT\_CMD); // COMMAND: assemble

ASM\_SRC\* parseASM(char\*); // parse ASM source

bool assemblerPass1(FILE\*, int\*, int\*); // Pass1 of SIC/XE Assembler

bool assemblerPass2(FILE\*, FILE\*, int, int); // Pass2 of SIC/XE Assembler

void printLST(FILE\*, ASM\_SRC\*, int, bool, bool); // create content into .lst file

void printOBJ(FILE\*, ASM\_SRC\*, int, int); // create content into .obj file

void printOBJList(FILE\*); // print object code list

void printModList(FILE\*); // print modification record list

void addOBJList(ASM\_SRC\*); // add node to object code list

void addModList(int, int); // add node to modification record list

void setError(ASM\_SRC\*, ASM\_ERROR); // mark error in ASM source

void printASMError(ASM\_SRC\*); // print detail about error

bool isRegister(char); // check whether operand is a register name

void symbolCMD(); // COMMAND: symbol

void symTableAdd(char\*, int); // add node to SYMTAB

SYMBOL\_ENTRY\* symTableSearch(char\*); // search for node in SYMTAB

## assembler.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: assembler.c \*

\* File description: Tasks for SIC/XE Assembler. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "assembler.h"

SYMBOL\_ENTRY\* symTable = NULL; // symbol table

ASM\_SRC\* parseList = NULL; // ASM source parse list

OBJ\_CODE\* objList = NULL; // object code listt

MOD\_RECORD\* modList = NULL; // modification record list

// ASM directives list

char directives[8][6] = {

"NOTDR",

"START",

"END",

"BASE",

"BYTE",

"WORD",

"RESB",

"RESW"

};

// ASM register list

char regs[7] = {

'A', 'X', 'L', 'B', 'S', 'T', 'F'

};

// COMMAND: assemble

bool assembleCMD(INPUT\_CMD ipcmd) {

FILE \*srcFile, \*lstFile, \*objFile;

char lstName[CMD\_LEN] = {'\0'}, objName[CMD\_LEN] = {'\0'};

int maxSrcLen = 0, progLen = 0;

// initialize

symTableFree();

parseListFree();

objListFree();

modListFree();

lstFile = objFile = NULL;

srcFile = fopen(ipcmd.arg[0], "r"); // open source file

if(!srcFile) {

puts("ERROR: File not found.");

return false;

}

// create file name for .lst and .obj files

strncpy(lstName, ipcmd.arg[0], strlen(ipcmd.arg[0]) - 3);

strncpy(objName, ipcmd.arg[0], strlen(ipcmd.arg[0]) - 3);

strcat(lstName, "lst");

strcat(objName, "obj");

lstFile = fopen(lstName, "w");

objFile = fopen(objName, "w");

if(!lstFile || !objFile) {

puts("ERROR: Problem while creating .list and .obj files.");

fclose(srcFile);

if(lstFile) fclose(lstFile);

if(objFile) fclose(objFile);

return false;

}

// both Pass1 and Pass2 have to be completed

if(!assemblerPass1(srcFile, &maxSrcLen, &progLen) || !assemblerPass2(lstFile, objFile, maxSrcLen, progLen)) {

puts(".lst file and .obj file were not created.");

remove(lstName);

remove(objName);

fclose(srcFile);

if(lstFile) fclose(lstFile);

if(objFile) fclose(objFile);

return false;

}

else

printf("\toutput file : [%s], [%s]\n", lstName, objName);

if(fclose(srcFile))

puts("WARNING: Error closing file.");

if(fclose(lstFile))

puts("WARNING: Error closing file.");

if(fclose(objFile))

puts("WARNING: Error closing file.");

return true;

}

// Pass1 of Assembler

bool assemblerPass1(FILE\* srcFile, int\* maxSrcLen, int\* progLen) {

ASM\_SRC \*curParse = NULL, \*prevParse = NULL;

char source[ASM\_LEN] = {'\0'};

int location = -1;

int lineNum = 5;

bool errorFlag = false;

int mult;

// process ASM source file line by line

while(fgets(source, ASM\_LEN, srcFile) != NULL) {

// remove '\n' at the end of string

if(source[strlen(source) - 1] == '\n')

source[strlen(source) - 1] = '\0';

curParse = parseASM(source); // get parsed line

// START directive found

if(curParse->direcName == START) {

// if START directive appears for the second time

if(location != -1) {

puts("ERROR: START directive appeared twice.");

errorFlag = true;

break;

}

// initialize LOCCTR

location = hexToDec(curParse->operand[0]);

curParse->location = location;

parseList = curParse;

prevParse = NULL;

}

// no START directive, set start address as 0

if(location == -1) {

location = 0;

parseList = curParse;

prevParse = NULL;

curParse->location = location;

}

else if(prevParse)

prevParse->next = curParse; // link node to list

curParse->lineNum = lineNum;

curParse->location = location;

// if current line is NOT a comment

if(curParse->type != COMMENT) {

// remember longest ASM source line

if(\*maxSrcLen < strlen(source))

\*maxSrcLen = strlen(source);

// if current line has symbol in label field

if(curParse->hasLabel && curParse->direcName != START) {

// if symbol is already in SYMTAB

if(symTableSearch(curParse->label)) {

setError(curParse, SYMBOL);

errorFlag = true;

break;

}

// new symbol found

else

symTableAdd(curParse->label, location);

}

switch(curParse->type) {

case ERROR: // Error

errorFlag = true;

break;

case INST: // Instruction

curParse->byteSize = curParse->format;

break;

case PSEUDO: // Pseudo-instruction

// directives

switch(curParse->direcName) {

case START:

break;

case END:

case BASE:

break;

case BYTE:

// calculate byte size according to operand

switch(curParse->operand[0][0]) {

case 'X':

curParse->byteSize = (strlen(curParse->operand[0]) - 3 + 1) / 2;

break;

case 'C':

curParse->byteSize = strlen(curParse->operand[0]) - 3;

break;

default:

mult = INC\_BYTE;

for(curParse->byteSize = 1;

mult <= atoi(curParse->operand[0]);

mult \*= INC\_BYTE)

curParse->byteSize++;

break;

}

break;

case WORD:

curParse->byteSize = 3;

break;

case RESB:

curParse->byteSize = atoi(curParse->operand[0]);

break;

case RESW:

curParse->byteSize = atoi(curParse->operand[0]) \* 3;

break;

default:

break;

}

break;

case COMMENT:

break;

default:

break;

}

}

if(errorFlag)

break;

location += curParse->byteSize; // calculate next location counter

lineNum += 5; // increase line number

prevParse = curParse; // previous node

// condition to end parsing

if(!strcmp(curParse->inst, "END"))

break;

}

\*progLen = location; // save the program length for .obj file

if(errorFlag) {

printASMError(curParse);

symTableFree();

return false;

}

return true;

}

// Pass2 of Assembler

bool assemblerPass2(FILE\* lstFile, FILE\* objFile, int maxSrcLen, int progLen) {

ASM\_SRC \*curParse = NULL;

SYMBOL\_ENTRY \*curSymbol = NULL;

int pcReg, baseReg;

bool errorFlag = false;

bool locFlag, objFlag;

bool nFlag, iFlag, xFlag, bFlag, pFlag, eFlag;

int i, firstExec;

// initialize

curParse = parseList;

baseReg = 0;

pcReg = parseList->location;

while(curParse) {

// initialize

locFlag = objFlag = true;

curSymbol = NULL;

// change PC register to next instruction

if(curParse->next)

pcReg = curParse->next->location;

// if no START directive at start, choose a blank program name

if(curParse == parseList && curParse->direcName != START) {

printf("H%6s%06X%06X\n", " ", 0, progLen);

firstExec = curParse->location;

}

switch(curParse->type) {

case INST: // instruction

// initialize

nFlag = iFlag = true;

xFlag = bFlag = pFlag = eFlag = false;

curParse->objCode = bucketSearch(curParse->inst + (curParse->inst[0] == '+' ? 1 : 0) )->codeVal;

switch(curParse->format) {

case format1:

break;

case format2:

// determine object code of each register

for(i = 0; i < 2; i++) {

curParse->objCode \*= 16;

switch(curParse->operand[i][0]) {

case 'A':

curParse->objCode += 0;

break;

case 'X':

curParse->objCode += 1;

break;

case 'L':

curParse->objCode += 2;

break;

case 'B':

curParse->objCode += 3;

break;

case 'S':

curParse->objCode += 4;

break;

case 'T':

curParse->objCode += 5;

break;

case 'F':

curParse->objCode += 6;

break;

default:

break;

}

}

break;

case format3:

if(!curParse->operandCnt) {

curParse->objCode += 3;

curParse->objCode \*= INC\_BYTE \* INC\_BYTE;

break;

}

// indexing mode

if(curParse->indexing)

xFlag = true;

// immediate addressing

if(curParse->operand[0][0] == '#')

nFlag = false;

// indirect addressing

else if(curParse->operand[0][0] == '@')

iFlag = false;

// if there is a symbol in operand

if((curSymbol = symTableSearch(curParse->operand[0] + ((!nFlag || !iFlag) ? 1 : 0)))) {

// if pc relative addressing is possible

if(curSymbol->address - pcReg + MAX12DISP / 2 >= 0) {

curParse->objCode += (!nFlag ? 1 : (!iFlag ? 2 : 3));

curParse->objCode \*= INC\_HBYTE;

curParse->objCode += (xFlag ? 10 : 2);

curParse->objCode \*= INC\_BYTE \* INC\_HBYTE;

curParse->objCode += (curSymbol->address - pcReg) & 0xFFF;

}

// if base relative addressing is possible

else if(curSymbol->address - baseReg < MAX12DISP) {

curParse->objCode += (!nFlag ? 1 : (!iFlag ? 2 : 3));

curParse->objCode \*= INC\_HBYTE;

curParse->objCode += (xFlag ? 12 : 4);

curParse->objCode \*= INC\_BYTE \* INC\_HBYTE;

curParse->objCode += (curSymbol->address - baseReg) & 0xFFF;

}

// approach as SIC instruction

else if(curSymbol->address < MAX15ADDR){

curParse->objCode \*= INC\_HBYTE;

curParse->objCode += (xFlag ? 8 : 0);

curParse->objCode \*= INC\_BYTE \* INC\_HBYTE;

curParse->objCode += curSymbol->address;

}

// NOT able to make reference

else {

setError(curParse, INSTRUCTION);

return false;

}

}

// if there is a value in operand

else if(pcReg - atoi(curParse->operand[0] + 1) + MAX12DISP / 2 >= 0) {

curParse->objCode += (!nFlag ? 1 : 2);

curParse->objCode \*= INC\_HBYTE;

curParse->objCode += 0;

curParse->objCode \*= INC\_BYTE \* INC\_HBYTE;

curParse->objCode += atoi(curParse->operand[0] + 1);

}

break;

case format4:

eFlag = true;

// search for symbol

curSymbol = symTableSearch(curParse->operand[0] + (!isalnum(curParse->operand[0][0]) ? 1 : 0));

// if there is a symbol in operand

if(curSymbol) {

curParse->objCode += 3;

curParse->objCode \*= INC\_HBYTE;

curParse->objCode += 1;

curParse->objCode \*= INC\_BYTE \* INC\_BYTE \* INC\_HBYTE;

curParse->objCode += curSymbol->address & 0xFFFFF;

}

// if there is a numeric value in operand

else {

curParse->objCode += 1;

curParse->objCode \*= INC\_HBYTE;

curParse->objCode += 1;

curParse->objCode \*= INC\_BYTE \* INC\_BYTE \* INC\_HBYTE;

curParse->objCode += atoi(curParse->operand[0] + 1) & 0xFFFFF;

}

break;

default:

break;

}

break;

case PSEUDO: // pseudo-instruction

switch(curParse->direcName) {

case START:

// initialize base register and remember first location

objFlag = false;

baseReg = curParse->location;

firstExec = curParse->location;

printOBJ(objFile, curParse, progLen, firstExec);

break;

case BASE:

// set base register

locFlag = objFlag = false;

baseReg = symTableSearch(curParse->operand[0])->address;

break;

case END:

locFlag = objFlag = false;

printOBJ(objFile, curParse, progLen, firstExec);

break;

case BYTE:

switch(curParse->operand[0][0]) {

case 'C': // a character constant

for(i = 2; curParse->operand[0][i] != '\''; i++) {

curParse->objCode \*= INC\_BYTE;

curParse->objCode += curParse->operand[0][i];

}

break;

case 'X': // a hexadecimal constant

curParse->objCode += strtol(curParse->operand[0] + 2, NULL, 16);

break;

default:

curParse->objCode = atoi(curParse->operand[0]);

break;

}

break;

case WORD:

curParse->objCode = atoi(curParse->operand[0]);

break;

case RESB:

case RESW:

objFlag = false;

break;

default:

break;

}

break;

case COMMENT:

locFlag = objFlag = false;

default:

break;

}

// if it has object code, print into .obj file

if(curParse->objCode)

printOBJ(objFile, curParse, progLen, firstExec);

// print into .lst file

printLST(lstFile, curParse, maxSrcLen, locFlag, objFlag);

curParse = curParse->next;

}

if(errorFlag) {

printASMError(curParse);

symTableFree();

return false;

}

return true;

}

void printLST(FILE\* lstFile, ASM\_SRC\* parsedASM, int maxSrcLen, bool printLOC, bool printOBJ) {

int i;

// print line number

fprintf(lstFile, "%4d\t", parsedASM->lineNum);

// print location if flag is set

(printLOC ? fprintf(lstFile, "%04X\t", parsedASM->location) : fprintf(lstFile, "\t"));

// print the source line with blanks filled to match the max length

for(i = 0; parsedASM->source[i]; i++)

fprintf(lstFile, "%c", parsedASM->source[i]);

for(;i < maxSrcLen; i++)

fprintf(lstFile, " ");

// print object code according to byte size

if(printOBJ) {

fprintf(lstFile, "\t\t");

switch(parsedASM->byteSize) {

case 1:

fprintf(lstFile, "%02X", parsedASM->objCode);

break;

case 2:

fprintf(lstFile, "%04X", parsedASM->objCode);

break;

case 3:

fprintf(lstFile, "%06X", parsedASM->objCode);

break;

case 4:

fprintf(lstFile, "%08X", parsedASM->objCode);

break;

default:

break;

}

}

fprintf(lstFile, "\n");

}

void printOBJ(FILE\* objFile, ASM\_SRC\* parsedASM, int progLen, int firstExec) {

static int lineLen = 0; // remember line lenght

// if START directive

if(parsedASM->direcName == START) {

fprintf(objFile, "H%-6s%06X%06X\n", parsedASM->label, parsedASM->location, progLen);

return;

}

// if END directive

if(parsedASM->direcName == END) {

printModList(objFile);

fprintf(objFile, "E%06X\n", firstExec);

return;

}

// if format 4 instruction with simple addressing add to modification record list

if(parsedASM->format == format4 && symTableSearch(parsedASM->operand[0] + (!isalnum(parsedASM->operand[0][0]) ? 1 : 0)))

addModList(parsedASM->location - firstExec + 1, 5);

// if line lenght exceeds 0x1E

if(lineLen + parsedASM->byteSize > 0x1E) {

fprintf(objFile, "T%06X%02X", objList ? objList->location : parsedASM->location, lineLen);

printOBJList(objFile);

objListFree();

lineLen = 0;

}

addOBJList(parsedASM);

lineLen += parsedASM->byteSize;

// if node was a variable, cut line

if(parsedASM->direcName == BYTE || parsedASM->direcName == WORD) {

fprintf(objFile, "T%06X%02X", objList ? objList->location : parsedASM->location, lineLen);

printOBJList(objFile);

objListFree();

lineLen = 0;

}

}

void printOBJList(FILE\* objFile) {

OBJ\_CODE\* cur = objList;

// print depending on byte size

while(cur) {

switch(cur->byteSize) {

case 1:

fprintf(objFile, "%02X", cur->objCode);

break;

case 2:

fprintf(objFile, "%04X", cur->objCode);

break;

case 3:

fprintf(objFile, "%06X", cur->objCode);

break;

case 4:

fprintf(objFile, "%08X", cur->objCode);

break;

default:

break;

}

cur = cur->next;

}

fprintf(objFile, "\n");

}

void printModList(FILE\* objFile) {

MOD\_RECORD\* cur = modList;

// print all modification records

while(cur) {

fprintf(objFile, "M%06X%02d\n", cur->location, cur->lenHB);

cur = cur->next;

}

}

void addOBJList(ASM\_SRC\* parsedASM) {

OBJ\_CODE \*newOBJ, \*cur = objList;

// create new node of object code list

newOBJ = (OBJ\_CODE\*) malloc(sizeof(OBJ\_CODE));

newOBJ->objCode = parsedASM->objCode;

newOBJ->location = parsedASM->location;

newOBJ->byteSize = parsedASM->byteSize;

newOBJ->next = NULL;

if(!objList) {

objList = newOBJ;

return;

}

while(cur->next)

cur = cur->next;

cur->next = newOBJ;

}

void addModList(int location, int length) {

MOD\_RECORD \*newRec, \*cur = modList;

// create new node of modification record list

newRec = (MOD\_RECORD\*) malloc(sizeof(MOD\_RECORD));

newRec->lenHB = length;

newRec->location = location;

newRec->next = NULL;

if(!modList) {

modList = newRec;

return;

}

while(cur->next)

cur = cur->next;

cur->next = newRec;

}

// free ASM source parse list

void parseListFree() {

ASM\_SRC \*cur, \*next;

cur = parseList;

while(cur) {

next = cur->next;

free(cur);

cur = next;

}

parseList = NULL;

}

// free object code list

void objListFree() {

OBJ\_CODE \*cur, \*next;

cur = objList;

while(cur) {

next = cur->next;

free(cur);

cur = next;

}

objList = NULL;

}

// free modification record list

void modListFree() {

MOD\_RECORD \*cur, \*next;

cur = modList;

while(cur) {

next = cur->next;

free(cur);

cur = next;

}

modList = NULL;

}

// print error detail of ASM source

void printASMError(ASM\_SRC\* parsedASM) {

printf("ERROR: Invalid assembly source.\n");

printf("[Line %d] Error in ", parsedASM->lineNum);

switch(parsedASM->errorCode) {

case SYMBOL:

puts("symbol field:");

break;

case INSTRUCTION:

puts("instruction field:");

break;

case OPERAND:

puts("operand field:");

break;

default:

break;

}

puts(parsedASM->source);

}

// COMMADN: symbol

void symbolCMD() {

SYMBOL\_ENTRY\* cur = symTable;

if(!symTable) {

puts("Symbol table is empty.");

return;

}

while(cur) {

printf("\t%s\t%04X\n", cur->symbol, cur->address);

cur = cur->next;

}

}

ASM\_SRC\* parseASM(char\* source) {

char delim[] = " \t\n";

char tmp[ASM\_LEN] = {'\0'};

char \*tok = NULL;

int i, j;

ASM\_SRC\* parseResult = NULL;

HASH\_ENTRY\* bucket = NULL;

// initialize parse structure

parseResult = (ASM\_SRC\*) malloc(sizeof(ASM\_SRC));

strcpy(parseResult->source, source);

memset(parseResult->label, '\0', ASM\_LEN);

memset(parseResult->inst, '\0', ASM\_LEN);

memset(parseResult->operand[0], '\0', ASM\_LEN);

memset(parseResult->operand[1], '\0', ASM\_LEN);

parseResult->hasLabel = false;

parseResult->indexing = false;

parseResult->operandCnt = 0;

parseResult->byteSize = 0;

parseResult->objCode = 0;

parseResult->type = INST;

parseResult->direcName = NOTDR;

parseResult->errorCode = OK;

parseResult->next = NULL;

// add space front and back of commas

j = 0;

for(i = 0; source[i]; i++) {

if(source[i] == ',') {

strcpy(tmp + j, " , ");

j += 3;

}

else

tmp[j++] = source[i];

}

tok = strtok(tmp, delim);

if(tok[0] == '.') { // comment found

parseResult->type = COMMENT;

parseResult->format = NONE;

return parseResult;

}

else if(!strcmp(tok, ",")) { // comma should not be present

setError(parseResult, INSTRUCTION);

return parseResult;

}

// look for directives

for(i = 1; i < 8; i++)

if(!strcmp(tok, directives[i])) { // pseudo instruction found

parseResult->type = PSEUDO;

parseResult->format = NONE;

parseResult->direcName = i;

strcpy(parseResult->inst, tok);

break;

}

if(parseResult->type != PSEUDO) { // if not a pseudo instruction

bucket = bucketSearch(tok + (tok[0] == '+' ? 1 : 0) );

// if first field is a label

if(!bucket) {

if(symTableSearch(tok)) { // symbol already exists

setError(parseResult, SYMBOL);

return parseResult;

}

else { // new symbol found, add to SYMTAB and move on

parseResult->hasLabel = true;

strcpy(parseResult->label, tok);

tok = strtok(NULL, delim);

// missing instruction

if(!tok) {

setError(parseResult, INSTRUCTION);

return parseResult;

}

bucket = bucketSearch(tok + (tok[0] == '+' ? 1 : 0) );

if(!strcmp(tok, ",")) { // comma should not be present

setError(parseResult, INSTRUCTION);

return parseResult;

}

}

}

// look for directives

for(i = 1; i < 8; i++)

if(!strcmp(tok, directives[i])) { // pseudo instruction found

parseResult->type = PSEUDO;

strcpy(parseResult->inst, tok);

parseResult->format = NONE;

parseResult->direcName = i;

break;

}

// if field is an instruction

if(bucket) {

parseResult->type = INST;

parseResult->format = bucket->format + 1; // get format from hash table

if(tok[0] == '+') {

// not a format 3/4 instruction

if(bucket->format != f34) {

setError(parseResult, INSTRUCTION);

return parseResult;

}

// found a format 4 instruction

parseResult->format = format4;

}

strcpy(parseResult->inst, tok);

}

// invalid instruction

else if(parseResult->type != PSEUDO){

setError(parseResult, INSTRUCTION);

return parseResult;

}

}

// tokenize the operand field part

tok = strtok(NULL, delim); // first operand

if(bucket && ((bucket->operandCnt && !tok) || (!bucket->operandCnt && tok))) {

setError(parseResult, OPERAND);

return parseResult;

}

else if(bucket && !bucket->operandCnt)

return parseResult;

if(tok && !strcmp(tok, ",")) { // comma should not be present

setError(parseResult, OPERAND);

return parseResult;

}

strcpy(parseResult->operand[0], tok);

parseResult->operandCnt = 1;

tok = strtok(NULL, delim); // expected a comma if there's second operand

if(tok && strcmp(tok, ",")) { // comma should be present

setError(parseResult, OPERAND);

return parseResult;

}

if(tok) {

tok = strtok(NULL, delim); // second operand

if(!tok || !strcmp(tok, ",")) {

setError(parseResult, OPERAND);

return parseResult;

}

strcpy(parseResult->operand[1], tok);

parseResult->operandCnt = 2;

}

// check operand format for instructions

if(parseResult->type == INST) {

switch(parseResult->format) {

case format2:

// first operand must be a Register

if(strlen(parseResult->operand[0]) != 1 || !isRegister(parseResult->operand[0][0])) {

setError(parseResult, OPERAND);

return parseResult;

}

// second operand for SHIFTL/R must be a number

if(!strncmp(parseResult->inst, "SHIFT", 5)) {

for(i = 0; parseResult->operand[1][i]; i++)

if(!isdigit(parseResult->operand[1][i])) {

setError(parseResult, OPERAND);

return parseResult;

}

}

// second operand must be a Register

else if(bucket->operandCnt != 1){

if(strlen(parseResult->operand[1]) != 1 || !isRegister(parseResult->operand[1][0])) {

setError(parseResult, OPERAND);

return parseResult;

}

}

break;

case format3:

case format4:

// one non-register operand or second operand is not X (indexing)

if((strlen(parseResult->operand[0]) == 1 && isRegister(parseResult->operand[0][0])) || (strlen(parseResult->operand[1]) && strcmp(parseResult->operand[1], "X")) ) {

setError(parseResult, OPERAND);

return parseResult;

}

if(strlen(parseResult->operand[1]))

parseResult->indexing = true;

break;

default:

break;

}

}

return parseResult;

}

// check for register name

bool isRegister(char reg) {

int i;

for(i = 0; i < 7; i++)

if(regs[i] == reg)

return true;

return false;

}

// set information about error in node

void setError(ASM\_SRC\* parsedResult, ASM\_ERROR error) {

parsedResult->type = ERROR;

parsedResult->format = NONE;

parsedResult->errorCode = error;

}

// add new symbol to SYMTAB

void symTableAdd(char\* symbol, int address) {

SYMBOL\_ENTRY\* cur = symTable;

SYMBOL\_ENTRY\* newEntry = (SYMBOL\_ENTRY\*) malloc(sizeof(SYMBOL\_ENTRY));

strcpy(newEntry->symbol, symbol);

newEntry->address = address;

newEntry->next = NULL;

// if SYMTAB is empty

if(!symTable) {

symTable = newEntry;

return;

}

// add in a non ascending order

if(strcmp(symTable->symbol, symbol) < 0) {

newEntry->next = symTable;

symTable = newEntry;

return;

}

// while next symbol is bigger in alphabetical order

while(cur->next && strcmp(cur->next->symbol, symbol) > 0)

cur = cur->next;

newEntry->next = cur->next;

cur->next = newEntry;

}

SYMBOL\_ENTRY\* symTableSearch(char\* symbol) {

SYMBOL\_ENTRY\* cur = symTable;

while(cur) {

// if a match is found

if(!strcmp(symbol, cur->symbol))

return cur;

cur = cur->next;

}

return NULL;

}

// free SYMTAB

void symTableFree() {

SYMBOL\_ENTRY \*cur, \*next;

cur = symTable;

while(cur) {

next = cur->next;

free(cur);

cur = next;

}

symTable = NULL;

}

## cmdProc.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: cmdProc.h \*

\* File description: Header file for input processing tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

INPUT\_CMD findCMD(char\*); // function to find command type of input

ERROR\_CODE testValidInput(INPUT\_CMD, COMMAND); // function to check validity of input command

void invFormatCMD(); // called when invalid format input

void invHexCMD(); // called when invalid hexadecimal value

void invValCMD(); // called when invalid value

void invFileCMD(); // called when invalid filename

## cmdProc.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: cmdProc.c \*

\* File description: Tasks related to parse input by user. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "cmdProc.h"

// store command format in structure array to compare

COMMAND cmdList[CMD\_CNT] = {

{ "help", "h", shell, help, false }, { "dir", "d", shell, dir, false },

{ "quit", "q", shell, quit, false }, { "history", "hi", shell, hist, false },

{ "dump", "du", memory, dump, true }, { "edit", "e", memory, edit, true },

{ "fill", "f", memory, fill, true }, { "reset", "reset", memory, reset, false },

{ "opcode", "opcode", opcode, op, true }, { "opcodelist", "opcodelist", opcode, oplist, false },

{ "type", "type", shell, type, true }, { "assemble", "assemble", assembler, assemble, true },

{ "symbol", "symbol", assembler, symbol, false }, { "progaddr", "progaddr", linkLoader, pAddr, true },

{ "loader", "loader", linkLoader, loader, true }, { "run", "run", linkLoader, run, false },

{ "bp", "bp", linkLoader, bp, true }, { "invalid", "invalid", invalid, invFormat, true }

};

INPUT\_CMD findCMD(char\* str) {

int i, j;

char delim[] = " \t\n"; // characters used to tokenize

char inp[CMD\_LEN];

char\* tok;

INPUT\_CMD ipcmd;

strcpy(inp, str); // copy input string

// initialize as invalid

ipcmd.cmd = invFormat;

ipcmd.argCnt = 0;

if(!strlen(str))

return ipcmd; // if empty string, return as invalid

tok = strtok(inp, delim); // first word of input

if(!tok)

return ipcmd; // emtpy token

for(i = 0; i < CMD\_CNT - 1; i++)

if(!strcmp(tok, cmdList[i].str) || !strcmp(tok, cmdList[i].abb)) {

ipcmd.cmd = cmdList[i].func; // if input command matches one of hard coded commands

break;

}

if(ipcmd.cmd == invFormat) // invalid command

return ipcmd;

// get arguments

j = 0;

while(tok) {

tok = strtok(NULL, delim); // next token (NOT expected comma if valid command)

if(ipcmd.cmd == loader && !tok)

break;

if(!j && !tok) // no argument for command

break;

if((j && !tok) || tok[0] == ',') { // there was a previous argument but empty token or comma found

ipcmd.cmd = invFormat; //invalid command

return ipcmd;

}

strcpy(ipcmd.arg[j++], tok); // copy argument to input command structure

if(ipcmd.cmd == loader)

continue;

tok = strtok(NULL, delim); // next token (expected a comma if valid command)

if(tok && tok[0] != ',') { // if token not empty, expected a comma

ipcmd.cmd = invFormat;

return ipcmd;

}

}

ipcmd.argCnt = j; // save argument count

// after input string parsed, do further check for validity

switch(testValidInput(ipcmd, cmdList[i])) { // get error type, if any

case FORMAT:

ipcmd.cmd = invFormat;

break;

case HEX:

ipcmd.cmd = invHex;

break;

case VALUE:

ipcmd.cmd = invVal;

break;

case FILENAME:

ipcmd.cmd = invFile;

break;

default: // no error found

break;

}

return ipcmd;

}

ERROR\_CODE testValidInput(INPUT\_CMD ipcmd, COMMAND format) {

int i;

int arg[3];

ERROR\_CODE code = SAFE; // initialize as correct command

if(ipcmd.cmd == invFormat)

return FORMAT;

// check argument count

switch(ipcmd.cmd) {

// strictly 0 arguments

case help:

case dir:

case quit:

case hist:

case reset:

case oplist:

case symbol:

case run:

if(ipcmd.argCnt)

code = FORMAT;

break;

// strictly 1 argument

case op:

case type:

case assemble:

case pAddr:

if(ipcmd.argCnt != 1)

code = FORMAT;

break;

// strictly 2 arguments

case edit:

if(ipcmd.argCnt != 2)

code = FORMAT;

break;

// strictly 3 arguments

case fill:

if(ipcmd.argCnt != 3)

code = FORMAT;

break;

// need less than 3 (0~2)

case dump:

if(ipcmd.argCnt > 2)

code = FORMAT;

break;

// need less than 2 (0~1)

case bp:

if(ipcmd.argCnt > 1)

code = FORMAT;

break;

// need at least 1 at most 3 (1~3)

case loader:

if(!ipcmd.argCnt || ipcmd.argCnt > 3)

code = FORMAT;

break;

default:

break;

}

if(code == FORMAT)

return code;

// check hexadecimal number if command is memory-related

if(format.category == memory) {

for(i = 0; i < ipcmd.argCnt; i++)

if((arg[i] = hexToDec(ipcmd.arg[i])) == -1)

code = HEX;

// check with each command's criteria

switch(ipcmd.cmd) {

case edit:

if(arg[0] >= MEM\_SIZE || arg[1] > 255)

code = VALUE;

break;

case fill:

if(arg[0] >= MEM\_SIZE || arg[1] >= MEM\_SIZE || arg[0] > arg[1] || arg[2] > 255)

code = VALUE;

break;

case dump:

switch(ipcmd.argCnt) {

case 2:

if(arg[1] >= MEM\_SIZE || arg[0] > arg[1])

code = VALUE;

case 1:

if(arg[0] >= MEM\_SIZE)

code = VALUE;

break;

}

break;

default:

break;

}

}

else if(format.category == assembler) {

if(ipcmd.cmd == assemble)

if(strcmp(ipcmd.arg[0] + strlen(ipcmd.arg[0]) - 4, ".asm"))

code = FILENAME;

}

else if(format.category == linkLoader) {

switch(ipcmd.cmd) {

case pAddr:

arg[0] = hexToDec(ipcmd.arg[0]);

if(arg[0] == -1)

code = HEX;

else if(arg[0] >= MEM\_SIZE)

code = VALUE;

break;

case bp:

arg[0] = hexToDec(ipcmd.arg[0]);

if(!strcmp(ipcmd.arg[0], "clear") || !ipcmd.argCnt)

break;

if(arg[0] == -1)

code = HEX;

if(arg[0] >= MEM\_SIZE)

code = VALUE;

break;

default:

break;

}

}

return code;

}

void invFormatCMD() {

puts("ERROR: Invalid command.");

puts("Type \"help\" for list and formats of commands.");

}

void invHexCMD() {

puts("ERROR: Incorrect hexadecimal.");

}

void invValCMD() {

puts("ERROR: Invalid address.");

puts("Memory size:\t\t1MB [0x00000 ~ 0xFFFFF]");

puts("Edit/Fill value range:\t 1B [0x00 ~ 0xFF]");

}

void invFileCMD() {

puts("ERROR: Invalid filename.");

puts("Assembly source file extension must be .asm");

}

int hexToDec(char\* hex) {

int i, dec = 0, multiplier = 1;

for(i = strlen(hex) - 1; i >= 0; i--) {

if(!isxdigit(hex[i])) // check if character if hexadecimal digit

return -1;

dec += multiplier \* (isdigit(hex[i]) ? (hex[i] - '0') : (toupper(hex[i]) - 'A' + 10));

multiplier \*= 16;

}

return dec;

}

## execute.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: execute.h \*

\* File description: Header file for execution related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

// add mnemonics as enum with corresponding opcode

typedef enum {

ADD = 0x18,

ADDF = 0x58,

ADDR = 0x90,

AND = 0x40,

CLEAR = 0xB4,

COMP = 0x28,

COMPF = 0x88,

COMPR = 0xA0,

DIV = 0x24,

DIVF = 0x64,

DIVR = 0x9C,

FIX = 0xC4,

FLOAT = 0xC0,

HIO = 0xF4,

J = 0x3C,

JEQ = 0x30,

JGT = 0x34,

JLT = 0x38,

JSUB = 0x48,

LDA = 0x00,

LDB = 0x68,

LDCH = 0x50,

LDF = 0x70,

LDL = 0x08,

LDS = 0x6C,

LDT = 0x74,

LDX = 0x04,

LPS = 0xD0,

MUL = 0x20,

MULF = 0x60,

MULR = 0x98,

NORM = 0xC8,

OR = 0x44,

RD = 0xD8,

RMO = 0xAC,

RSUB = 0x4C,

SHIFTL = 0xA4,

SIO = 0xF0,

SSK = 0xEC,

STA = 0x0C,

STB = 0x78,

STCH = 0x54,

STF = 0x80,

STI = 0xD4,

STL = 0x14,

STS = 0x7C,

STSW = 0xE8,

STT = 0x84,

STX = 0x10,

SUB = 0x1C,

SUBF = 0x5C,

SUBR = 0x94,

SVC = 0xB0,

TD = 0xE0,

TIO = 0xF8,

TIX = 0x2C,

TIXR = 0xB8,

WD = 0xDC

} OPmnemonic;

// instruction format

typedef enum {

constant, fmt1, fmt2, fmt3, fmt4

} FMT;

// addressing mode

typedef enum {

SIC, immediate, indirect, simple

} ADR\_MODE;

// CC status ( <=, ==, >= )

enum {

lt, eq, gt

} CCstatus;

// struct to store parsed obj code

typedef struct {

OPmnemonic mnemonic;

FMT format;

ADR\_MODE addrMode;

union {

int target;

int immediate;

REG reg[2];

} operand;

bool indexing;

} OBJ;

bool bpCMD(INPUT\_CMD); // search for break point

bool searchBP(int); // search for break point

void runCMD(); // run command

void dumpReg(); // dump registers

int getTargetAddress(int, FMT); // return target address

int SICAddress(int); // SIC instruction

int immediateAddress(int, FMT); // immediate address

int indirectAddress(int, FMT); // indirect address

int simpleAddress(int, FMT); // simple address

int getMem(int, int); // get value from memory

void putMem(int, int, int); // store value in memory

## execute.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.h \*

\* File description: Performs the execution of programs. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "memory.h"

#include "linkedList.h"

#include "execute.h"

char inputStream[12] = " SIC/XE\0\0"; // virtual input device for testing copy.obj

char outputStream[13] = {'\0'}; // virtual output device for testing copy.obj

int inputIdx = 0;

int outputIdx = 0;

bool bpCMD(INPUT\_CMD ipcmd) {

BREAK\_PNT\* newBP;

// if no argument was in input

if(!ipcmd.argCnt) {

printf("\tbreakpoint\n");

printf("\t----------\n");

printList(breakPntList, printBreakPntList);

return true;

}

// if break clear was in input

if(!strcmp(ipcmd.arg[0], "clear")) {

printf("\t[ok] clear all breakpoints\n");

freeList(&breakPntList);

return true;

}

// if break point was already present at same address

if(searchBP(hexToDec(ipcmd.arg[0]))) {

printf("\t[warning] breakpoint already at %04X\n", hexToDec(ipcmd.arg[0]));

return false;

}

// add new break point to linked list

newBP = (BREAK\_PNT\*) malloc(sizeof(BREAK\_PNT));

newBP->address = hexToDec(ipcmd.arg[0]);

addToList(&breakPntList, (void\*) newBP);

printf("\t[ok] create breakpoint %04X\n", hexToDec(ipcmd.arg[0]));

return true;

}

bool searchBP(int address) {

NODE\* curBP = breakPntList;

while(curBP) {

if(((BREAK\_PNT\*)(curBP->data))->address == address)

return true; // break point was found

curBP = curBP->next;

}

return false;

}

void runCMD() {

static int curAddress = -1;

static int lastBP = -1;

int i;

int targetVal, targetAddress;

OBJ curObj;

// initialization

CCstatus = 4;

if(curAddress == -1)

curAddress = execAddress; // start from beginning

registers[PCreg] = curAddress; // set PC register

for(; curAddress < endAddress; ) {

curObj.mnemonic = mem[curAddress] & 0xFC; // decode opcode

// check instruction format

switch(mem[curAddress] / 0x10) {

case 0:

case 1:

case 2:

case 3:

case 4:

case 5:

case 6:

case 7:

case 8:

case 0xD:

case 0xE:

curObj.format = fmt3; // format 3

if(mem[curAddress + 1] & 0x10) // e bit is set

curObj.format = fmt4; // format 4

break;

case 9:

case 0xA:

case 0xB:

curObj.mnemonic += mem[curAddress] & 0x3;

curObj.format = fmt2; // format 2

break;

case 0xC:

case 0xF:

curObj.mnemonic += mem[curAddress] & 0x3;

curObj.format = fmt1; // format 1

break;

default:

break;

}

if(!mem[curAddress])

curObj.format = 1;

// check for breaK point

for(i = curAddress; i < curAddress + curObj.format; i++)

if(i > lastBP && searchBP(i)) {

lastBP = i;

dumpReg();

printf("\n\tStop at checkpoint[%04X]\n", i);

return;

}

// skip empty memory

if(!mem[curAddress]) {

registers[PCreg] += 1; // increase PC

curAddress = registers[PCreg];

continue;

}

registers[PCreg] += curObj.format; // increase PC

// get target address

curObj.addrMode = simple; // initialize as simple addressing mode

switch(curObj.format) {

case fmt3:

case fmt4:

curObj.operand.target = getTargetAddress(curAddress, curObj.format); // get target address

curObj.addrMode = mem[curAddress] & 3; // find addressing mode

if(mem[curAddress] & 0x80) // indexing mode

curObj.operand.target += registers[Xreg];

if(curObj.addrMode == immediate) // if immediate mode

curObj.operand.immediate = curObj.operand.target;

break;

case fmt2:

curObj.operand.reg[0] = mem[curAddress + 1] / 0x10; // decode register1

curObj.operand.reg[1] = mem[curAddress + 1] % 0x10; // decode register2

break;

case fmt1:

break;

default:

break;

}

targetAddress = curObj.operand.target;

if(curObj.operand.target < MEM\_SIZE)

targetVal = (curObj.addrMode == immediate ? curObj.operand.immediate : getMem(curObj.operand.target, 6));

// execute instruction

switch(curObj.mnemonic) {

case ADD:

registers[Areg] += targetVal;

break;

case ADDF:

registers[Freg] += targetVal;

break;

case ADDR:

registers[curObj.operand.reg[1]] += registers[curObj.operand.reg[0]];

break;

case AND:

registers[Areg] &= targetVal;

break;

case CLEAR:

registers[curObj.operand.reg[0]] = 0;

break;

case COMP:

if(registers[Areg] < targetVal)

CCstatus = lt;

else if(registers[Areg] == targetVal)

CCstatus = eq;

else

CCstatus = gt;

break;

case COMPF:

if(registers[Freg] < targetVal)

CCstatus = lt;

else if(registers[Freg] == targetVal)

CCstatus = eq;

else

CCstatus = gt;

break;

case COMPR:

if(registers[curObj.operand.reg[0]] < registers[curObj.operand.reg[1]])

CCstatus = lt;

else if(registers[curObj.operand.reg[0]] == registers[curObj.operand.reg[1]])

CCstatus = eq;

else

CCstatus = gt;

break;

case DIV:

if(!targetVal) {

puts("ERROR: division by 0. Program will end.");

curAddress = endAddress;

continue;

}

registers[Areg] /= targetVal;

break;

case DIVF:

if(!targetVal) {

puts("ERROR: division by 0. Program will end.");

curAddress = endAddress;

continue;

}

registers[Freg] /= targetVal;

break;

case DIVR:

if(!registers[curObj.operand.reg[0]]) {

puts("ERROR: division by 0. Program will end.");

curAddress = endAddress;

continue;

}

registers[curObj.operand.reg[1]] /= registers[curObj.operand.reg[0]];

break;

case FIX:

registers[Areg] = registers[Freg];

break;

case FLOAT:

registers[Freg] = registers[Areg];

break;

case HIO:

break;

case J:

registers[PCreg] = targetAddress;

break;

case JEQ:

if(CCstatus == eq)

registers[PCreg] = targetAddress;

break;

case JGT:

if(CCstatus == gt)

registers[PCreg] = targetAddress;

break;

case JLT:

if(CCstatus == lt)

registers[PCreg] = targetAddress;

break;

case JSUB:

registers[Lreg] = registers[PCreg];

registers[PCreg] = targetAddress;

break;

case LDA:

registers[Areg] = targetVal;

break;

case LDB:

registers[Breg] = targetVal;

break;

case LDCH: // load to lower byte

registers[Areg] = (registers[Areg] & 0xFFFFFF00) + (targetVal / 0x10000);

break;

case LDF:

registers[Freg] = targetVal;

break;

case LDL:

registers[Lreg] = targetVal;

break;

case LDS:

registers[Sreg] = targetVal;

break;

case LDT:

registers[Treg] = targetVal;

break;

case LDX:

registers[Xreg] = targetVal;

break;

case LPS:

break;

case MUL:

registers[Areg] \*= targetVal;

break;

case MULF:

registers[Freg] \*= targetVal;

break;

case MULR:

registers[curObj.operand.reg[1]] \*= registers[curObj.operand.reg[0]];

break;

case NORM:

break;

case OR:

registers[Areg] |= targetVal;

break;

case RD: // read to lower byte

registers[Areg] = (registers[Areg] & 0xFFFFFF00) + inputStream[inputIdx++];

break;

case RMO:

registers[curObj.operand.reg[1]] = registers[curObj.operand.reg[0]];

break;

case RSUB:

registers[PCreg] = registers[Lreg];

break;

case SHIFTL:

registers[curObj.operand.reg[0]] = registers[curObj.operand.reg[0]] << registers[curObj.operand.reg[1]];

break;

case SIO:

break;

case SSK:

break;

case STA:

putMem(targetAddress, 3, registers[Areg]);

break;

case STB:

putMem(targetAddress, 3, registers[Breg]);

break;

case STCH: // store lower byte

mem[targetAddress] = registers[Areg] & 0xFF;

break;

case STF:

putMem(targetAddress, 6, registers[Freg]);

break;

case STI:

break;

case STL:

putMem(targetAddress, 3, registers[Lreg]);

break;

case STS:

putMem(targetAddress, 3, registers[Sreg]);

break;

case STSW:

putMem(targetAddress, 3, registers[SWreg]);

break;

case STT:

putMem(targetAddress, 3, registers[Treg]);

break;

case STX:

putMem(targetAddress, 3, registers[Xreg]);

break;

case SUB:

registers[Areg] -= targetVal;

break;

case SUBF:

registers[Freg] -= targetVal;

break;

case SUBR:

registers[curObj.operand.reg[1]] -= registers[curObj.operand.reg[0]];

break;

case SVC:

break;

case TD:

CCstatus = lt; // suppose it's always active

break;

case TIO:

CCstatus = lt;

break;

case TIX:

registers[Xreg]++;

if(registers[Xreg] < targetVal)

CCstatus = lt;

else if(registers[Xreg] == targetVal)

CCstatus = eq;

else

CCstatus = gt;

break;

case TIXR:

registers[Xreg]++;

if(registers[Xreg] < registers[curObj.operand.reg[0]])

CCstatus = lt;

else if(registers[Xreg] == registers[curObj.operand.reg[0]])

CCstatus = eq;

else

CCstatus = gt;

break;

case WD: // store lower byte

outputStream[outputIdx++] = registers[Areg] & 0xFF;

break;

default: // not an opcode

registers[PCreg] = registers[PCreg] - curObj.format + 1; // increase PC

curAddress = registers[PCreg];

continue;

break;

}

curAddress = registers[PCreg]; // increase curAddress

}

// program ended, dump registers

registers[PCreg] = endAddress;

dumpReg();

printf("\n\tEnd program.\n");

// prepare for next run

for(i = 0; i < REG\_CNT; i++)

registers[i] = 0;

registers[Lreg] = endAddress;

curAddress = -1;

lastBP = -1;

CCstatus = 4;

inputIdx = outputIdx = 0;

memset(outputStream, '\0', 13);

}

int getTargetAddress(int curAddress, FMT format) {

ADR\_MODE addrMode = mem[curAddress] & 3; // mask on n and i bits

int target;

switch(addrMode) {

case SIC: // n = 0, i = 0

target = SICAddress(curAddress);

break;

case immediate: // n = 0, i = 0

target = immediateAddress(curAddress, format);

break;

case indirect: // n = 1, i = 0

target = indirectAddress(curAddress, format);

break;

case simple: // n = 1, i = 1

target = simpleAddress(curAddress, format);

break;

default:

break;

}

if(mem[curAddress + 1] & 0x80) // indexing mode

target += registers[Xreg];

return target;

}

int SICAddress(int curAddress) {

return getMem(curAddress, 5) & 0x7FFF; // lower 15 bits is target

}

int immediateAddress(int curAddress, FMT format) {

return simpleAddress(curAddress, format);

}

int indirectAddress(int curAddress, FMT format) {

int target = simpleAddress(curAddress, format);

return getMem(target, 6);

}

int simpleAddress(int curAddress, FMT format) {

int setBit = (mem[curAddress + 1] / 0x10) & 6; // mask over b and p bits

int target = getMem(curAddress + 1, format == fmt3 ? 3 : 5);

if(setBit == 2) { // PC relative

if(target & (format == fmt3 ? 0x800 : 0x80000)) // mask over displacement field

target = target | (format == fmt3 ? 0xFFFFF000 : 0xFFF00000);

target += registers[PCreg];

}

else if(setBit == 4) // Base relative

target += registers[Breg];

return target;

}

int getMem(int address, int hBytes) {

int val = 0;

int i;

if(address >= MEM\_SIZE)

return 0;

val = mem[address] % (hBytes % 2 ? 0x10 : 0x100); // get half or full byte

for(i = 1; i <= (hBytes - 1) / 2; i++) {

val \*= 0x100; // increae byte

val += mem[address + i];

}

return val;

}

void putMem(int address, int bytes, int value) {

int i;

if(address >= MEM\_SIZE)

return;

for(i = address + bytes - 1; i >= address; i--) {

mem[i] = value & 0xFF; // mask over lower byte

value /= 0x100;

}

}

// output registers

void dumpReg() {

printf("\t A : %012X X : %08X\n", registers[Areg], registers[Xreg]);

printf("\t L : %012X PC: %012X\n", registers[Lreg], registers[PCreg]);

printf("\t B : %012X S : %012X\n", registers[Breg], registers[Sreg]);

printf("\t T : %012X", registers[Treg]);

}

## hash.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.h \*

\* File description: Header file for hash table related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#define OPCODE\_LEN 7

// HASH\_ENTRY\* hashTable[HASH\_SIZE]; // hash table pointer array

void opCMD(INPUT\_CMD); // COMMAND: opcode

void oplistCMD(); // COMMAND: opcodelist

void hashCreate(); // function to create hash table

void checkOperandCnt(void\*);

void hashAddBucket(int, void\*); // function to add bucket to hash table

int hashFunction(char\*); // function that returns hash function

## hash.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.c \*

\* File description: Tasks handling opcode and hash table. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "linkedList.h"

#include "hash.h"

// Format 2 instruction but 1 operand

char exceptionFmt2[][OPCODE\_LEN] = {

"CLEAR",

"SVC",

"TIXR"

};

// Format 3 instruction but NO operand

char exceptionFmt3[][OPCODE\_LEN] = {

"RSUB"

};

void opCMD(INPUT\_CMD ipcmd) {

HASH\_ENTRY\* bucket = bucketSearch(ipcmd.arg[0]);

if(bucket) // target found!

printf("opcode is %s\n", bucket->code);

else { // target NOT found..

puts("ERROR: mnemonic not found.");

puts("Type \"opcodelist\" for list of available opcodes.");

}

}

void oplistCMD() {

NODE\* bucket;

int i;

for(i = 0; i < HASH\_SIZE; i++) {

printf("%2d : ", i); // print table index

bucket = opCodeTable[i]; // get front bucket

printList(bucket, printOpList);

/\*

while(bucket) {

printf("[%s,%s]", bucket->inst, bucket->code);

if((bucket = bucket->next)) // if there exists next bucket

printf(" -> ");

}

\*/

puts("");

}

}

void hashCreate() {

FILE\* fp = fopen("opcode.txt", "r"); // open file

char cd[3], ins[10], md[4];

void\* bucket;

int hash;

if(!fp) {

puts("ERROR: Unable to load \"opcode.txt\".");

return;

}

while(fscanf(fp, "%s %s %s", cd, ins, md) == 3) {

// allocate and fill in new data

bucket = malloc(sizeof(HASH\_ENTRY));

strcpy(((HASH\_ENTRY\*)bucket)->code, cd);

strcpy(((HASH\_ENTRY\*)bucket)->inst, ins);

((HASH\_ENTRY\*)bucket)->codeVal = hexToDec(cd);

((HASH\_ENTRY\*)bucket)->format = md[0] - '1';

checkOperandCnt(bucket);

((HASH\_ENTRY\*)bucket)->next = NULL;

// calculate hash

hash = hashFunction(((HASH\_ENTRY\*)bucket)->inst);

// add to generic list

addToList(opCodeTable + hash, bucket);

// add to internal list for print issues

hashAddBucket(hash, bucket);

}

if(fclose(fp)) {

puts("WARNING: Error closing \"opcode.txt\".");

return;

}

}

void checkOperandCnt(void\* bucket) {

int i;

switch(((HASH\_ENTRY\*)bucket)->format) {

case f1: // Format 1 instruction

((HASH\_ENTRY\*)bucket)->operandCnt = 0;

break;

case f2: // Format 2 instruction

((HASH\_ENTRY\*)bucket)->operandCnt = 2; // Default is 2 operands

for(i = 0; i < 3; i++) // Exception check (CLEAR, SVC, TIXR)

if(!strcmp(((HASH\_ENTRY\*)bucket)->inst, exceptionFmt2[i])) {

((HASH\_ENTRY\*)bucket)->operandCnt = 1;

break;

}

break;

case f34: // Format 3 or 4 instruction

((HASH\_ENTRY\*)bucket)->operandCnt = 1; // Default is 1 operand

for(i = 0; i < 1; i++) // Exception check (RSUB)

if(!strcmp(((HASH\_ENTRY\*)bucket)->inst, exceptionFmt3[i])) {

((HASH\_ENTRY\*)bucket)->operandCnt = 0;

break;

}

break;

}

}

int hashFunction(char\* inst) {

return abs( (int) inst[0] \* 2 + abs(inst[0] + inst[1] + inst[2]) ) % HASH\_SIZE;

}

void hashAddBucket(int hash, void\* bucket) {

HASH\_ENTRY\* cur = (HASH\_ENTRY\*)(opCodeTable[hash]->data);

if(!strcmp(cur->inst, ((HASH\_ENTRY\*)bucket)->inst)) // if it is the first bucket

return;

while(cur->next)

cur = cur->next; // go to the end of list

cur->next = (HASH\_ENTRY\*)bucket;

}

HASH\_ENTRY\* bucketSearch(char\* inst) {

void\* bucket;

char tmp[ASM\_LEN] = {'\0'};

strcpy(tmp, inst);

bucket = opCodeTable[hashFunction(tmp)]->data; // get front bucket from hash function

while(bucket && strcmp(((HASH\_ENTRY\*)bucket)->inst, inst)) // search till match or end

bucket = ((HASH\_ENTRY\*)bucket)->next;

return (HASH\_ENTRY\*)bucket;

}

## linkedList.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: linkedList.h \*

\* File description: Header file for linked list related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

LIST histList;

LIST opCodeTable[HASH\_SIZE];

LIST extSymTable;

LIST breakPntList;

void addToList(LIST\*, void\*);

void freeList(LIST\*);

void opCodeTableFree();

void extSymTableFree();

void printList(LIST, void (void\*));

void printHistory(void\*);

void printOpList(void\*);

void printCntSecTable(void\*);

void printExtSym(void\*);

void printBreakPntList(void\*);

## linkedList.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: linkedList.c \*

\* File description: Manages functions related to linked lists. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "linkedList.h"

void addToList(LIST\* listHead, void\* data) {

NODE\* cur = \*listHead;

NODE\* newNode = malloc(sizeof(NODE));

newNode->data = data;

newNode->next = NULL;

if(!cur) {

\*listHead = newNode;

return;

}

while(cur->next)

cur = cur->next;

cur->next = newNode;

}

void freeList(LIST\* listHead) {

NODE\* cur = \*listHead;

NODE\* next = NULL;

while(cur) {

next = cur->next;

free(cur->data);

free(cur);

cur = next;

}

\*listHead = NULL;

}

void opCodeTableFree() {

int i;

for(i = 0; i < HASH\_SIZE; i++)

freeList(opCodeTable + i);

}

void extSymTableFree() {

NODE\* cur = extSymTable;

while(cur) {

freeList(&(((CNT\_SEC\*)(cur->data))->extSym));

cur = cur->next;

}

freeList(&extSymTable);

}

void printList(LIST listHead, void (fptr)(void\*)) {

NODE\* cur = listHead;

while(cur) {

(fptr)(cur->data);

cur = cur->next;

}

}

void printHistory(void\* data) {

static int index = 1;

if(data == histList->data)

index = 1;

printf("%-3d ", index++);

puts(((HIST\_NODE\*)data)->str);

}

void printOpList(void\* data) {

printf("[%s,%s]", ((HASH\_ENTRY\*)data)->inst, ((HASH\_ENTRY\*)data)->code);

if(((HASH\_ENTRY\*)data)->next)

printf(" -> ");

}

void printCntSecTable(void\* data) {

if(data == extSymTable->data) {

printf("Control\t\tSymbol\t\tAddress\t\tLength\n");

printf("section\t\tname\n");

printf("---------------------------------------------------------\n");

}

printf("%s\t\t\t\t%04X\t\t%04X\n", ((CNT\_SEC\*)data)->csName, ((CNT\_SEC\*)data)->stAddress, ((CNT\_SEC\*)data)->length);

printList(((CNT\_SEC\*)data)->extSym, printExtSym);

}

void printExtSym(void\* data) {

printf("\t\t%s\t\t%04X\n", ((EXT\_SYMBOL\*)data)->symName, ((EXT\_SYMBOL\*)data)->address);

}

void printBreakPntList(void\* data) {

printf("\t%04X\n", ((BREAK\_PNT\*)data)->address);

}

## linkLoader.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.h \*

\* File description: Header file for linking loaderd tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

void pAddrCMD(INPUT\_CMD); // progaddr COMMAND

bool loaderCMD(INPUT\_CMD); // loader COMMAND

int linkLoaderPass1(FILE\*\*); // Pass1 of linking loader

int linkLoaderPass2(FILE\*\*); // Pass2 of linking loader

void fcloseObj(FILE\*\*); // close obj file pointers

bool searchCS(char\*); // search for control section in ESTAB

int searchES(char\*); // search for external symbol in ESTAB

## linkLoader.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: hash.h \*

\* File description: Performs the linking loader task. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "linkedList.h"

#include "memory.h"

#include "linkLoader.h"

// set program start address input by user

void pAddrCMD(INPUT\_CMD ipcmd) {

progAddr = execAddress = strtol(ipcmd.arg[0], NULL, 16);

}

// linking loader process

bool loaderCMD(INPUT\_CMD ipcmd) {

FILE\* objFptr[3];

int progLen;

int i;

objFptr[0] = objFptr[1] = objFptr[2] = NULL;

extSymTableFree(); // initialize ESTAB

for(i = 0; i < REG\_CNT; i++)

registers[i] = 0; // initialize registers

// check for file error

for(i = 0; i < ipcmd.argCnt; i++) {

if(strcmp(ipcmd.arg[i] + strlen(ipcmd.arg[i]) - 4, ".obj")) {

puts("ERROR: Non-object file selected.");

fcloseObj(objFptr);

return false;

}

if(!(objFptr[i] = fopen(ipcmd.arg[i], "r"))) {

printf("ERROR: %s not found.\n", ipcmd.arg[i]);

fcloseObj(objFptr);

return false;

}

}

progLen = linkLoaderPass1(objFptr); // pass 1 of linking loader

if(!progLen)

return false;

for(i = 0; i < 3; i++)

if(objFptr[i])

rewind(objFptr[i]); // rewind file stream to beginning

execAddress = linkLoaderPass2(objFptr); // pass 2 of linking loader

if(execAddress == -1)

return false;

registers[Lreg] = progAddr + progLen; // set L register to program's end address

printList(extSymTable, printCntSecTable); // print load map

printf("---------------------------------------------------------\n");

printf("\t\t\t\tTotal length\t%04X\n", progLen);

fcloseObj(objFptr);

return true;

}

int linkLoaderPass1(FILE\*\* objFptr) {

int i = 0, j;

int CSADDR = 0;

char record[101];

char csName[CS\_LEN], esName[CS\_LEN], addr[CS\_LEN];

CNT\_SEC\* newCntSec = NULL;

EXT\_SYMBOL\* newExtSym = NULL;

newCntSec = (CNT\_SEC\*) malloc(sizeof(CNT\_SEC));

newCntSec->stAddress = CSADDR = progAddr; // start address of first control section

newCntSec->extSym = NULL;

while(objFptr[i]) {

fgets(record, 101, objFptr[i]); // header record

if(record[strlen(record) - 1] == '\n')

record[strlen(record) - 1] = '\0';

newCntSec->length = hexToDec(record + strlen(record) - 6); // control section length

strncpy(csName, record + 1, CS\_LEN - 1); // get control section name

if(record[0] != 'H') { // obj file is missing header record

printf("ERROR: missing header record in .obj file number %d.\n", i + 1);

extSymTableFree();

free(newCntSec);

return 0;

}

if(searchCS(csName)) { // same name for multiple control sections

puts("ERROR: overlapping control section name.");

extSymTableFree();

free(newCntSec);

return 0;

}

else { // add new control section to ESTAB

strcpy(newCntSec->csName, csName);

addToList(&extSymTable, (void\*) newCntSec);

}

fgets(record, 101, objFptr[i]); // read next record

if(record[strlen(record) - 1] == '\n')

record[strlen(record) - 1] = '\0';

while(record[0] != 'E') {

if(record[0] == 'D') { // if a D record is found

for(j = 1; record[j]; j += 12) {

strncpy(esName, record + j, CS\_LEN - 1); // get symbol name

strncpy(addr, record + j + 6, CS\_LEN - 1); // get symbol address

if(searchES(esName) != -1) { // same name for multiple symbols

puts("ERROR: overlapping symbol name.");

extSymTableFree();

return 0;

}

newExtSym = (EXT\_SYMBOL\*) malloc(sizeof(EXT\_SYMBOL));

strncpy(newExtSym->symName, esName, CS\_LEN - 1);

newExtSym->address = hexToDec(addr) + CSADDR;

addToList(&(newCntSec->extSym), (void\*) newExtSym); // add new external symbol to ESTAB

}

}

fgets(record, 101, objFptr[i]); // read next record

if(record[strlen(record) - 1] == '\n')

record[strlen(record) - 1] = '\0';

}

CSADDR += newCntSec->length; // increment CSADDR

i++;

if(i == CS\_MAX) // maximum of 3 control sections

break;

if(objFptr[i]) { // next control section exists

newCntSec = (CNT\_SEC\*) malloc(sizeof(CNT\_SEC));

newCntSec->stAddress = CSADDR;

newCntSec->extSym = NULL;

}

}

endAddress = CSADDR; // set program's end address

return CSADDR - progAddr;

}

int linkLoaderPass2(FILE\*\* objFptr) {

int i = 0, j, k;

int offset, tLen, hByteCnt, modAddress, maxExSymIndex = -1;

int CSADDR = 0, EXECADDR = 0, CSLTH = 0;

int\* refNum = NULL;

char record[101];

char esName[CS\_LEN], temp[CS\_LEN];

NODE\* curCntSec = extSymTable;

CSADDR = EXECADDR = progAddr; // start address of first control section

while(objFptr[i]) {

fgets(record, 101, objFptr[i]); // header record

if(record[strlen(record) - 1] == '\n')

record[strlen(record) - 1] = '\0';

CSLTH = ((CNT\_SEC\*)(curCntSec->data))->length;

fgets(record, 101, objFptr[i]); // read next record

if(record[strlen(record) - 1] == '\n')

record[strlen(record) - 1] = '\0';

while(record[0] != 'E') {

if(record[0] == 'T') { // if a T record is found

memset(temp, '\0', CS\_LEN);

strncpy(temp, record + 1, CS\_LEN - 1);

offset = hexToDec(temp); // text record start address offset

memset(temp, '\0', CS\_LEN);

strncpy(temp, record + 7, 2);

tLen = hexToDec(temp); // text record length

for(j = 0; j < tLen; j++) {

memset(temp, '\0', CS\_LEN);

strncpy(temp, record + 9 + j \* 2, 2);

mem[CSADDR + offset + j] = hexToDec(temp); // load to memory

}

}

else if(record[0] == 'M') { // if M record is found

memset(temp, '\0', CS\_LEN);

strncpy(temp, record + 1, CS\_LEN - 1);

offset = hexToDec(temp); // byte location to modify

memset(temp, '\0', CS\_LEN);

strncpy(temp, record + 7, 2);

hByteCnt = hexToDec(temp); // half byte count to modify

memset(temp, '\0', CS\_LEN);

strcpy(temp, record + 10); // reference number

if(hexToDec(temp) > maxExSymIndex) {

puts("ERROR: out-of-bound reference number in modification record.");

return -1;

}

modAddress = mem[CSADDR + offset] % (hByteCnt % 2 ? 0x10 : 0x100); // first (half)byte to modify

for(j = 1; j <= (hByteCnt - 1) / 2; j++) {

modAddress \*= 0x100;

modAddress += mem[CSADDR + offset + j]; // the rest (half)byte to modify

}

modAddress += (record[9] == '+' ? 1 : -1) \* refNum[hexToDec(temp)]; // modification address

for(j = (hByteCnt - 1) / 2; j; j--) {

mem[CSADDR + offset + j] = modAddress % 0x100; // store back to memory

modAddress /= 0x100;

}

mem[CSADDR + offset] = (hByteCnt % 2 ? (mem[CSADDR + offset] / 0x10) \* 0x10 + modAddress : modAddress);

}

else if(record[0] == 'R') { // if R record is found

if(refNum)

free(refNum);

refNum = (int\*) malloc(sizeof(int) \* ((strlen(record) - 1) / 8 + 3));

for(j = 2; j < (strlen(record) - 1) / 8 + 3; j++) {

strncpy(esName, record + (j - 2) \* 8 + 3, CS\_LEN - 1);

for(k = strlen(esName); k < CS\_LEN - 1; k++) // fill with ' ' till 6 characters

esName[k] = ' ';

refNum[j] = searchES(esName);

}

maxExSymIndex = j - 1; // save max index for error checking

refNum[1] = CSADDR;

}

fgets(record, 101, objFptr[i]); // read next record

if(record[strlen(record) - 1] == '\n')

record[strlen(record) - 1] = '\0';

}

CSADDR += CSLTH; // increment CSADDR

i++;

if(i == CS\_MAX) // maximum of 3 control sections

break;

curCntSec = curCntSec->next; // next control section

}

if(refNum)

free(refNum);

return EXECADDR;

}

void fcloseObj(FILE\*\* objFptr) {

int i;

for(i = 0; i < 3; i++) {

if(objFptr[i])

fclose(objFptr[i]); // close each obj file

objFptr[i] = NULL;

}

}

bool searchCS(char\* csName) {

NODE\* cur = extSymTable;

CNT\_SEC\* data;

while(cur) {

data = (CNT\_SEC\*) cur->data;

if(!strcmp(data->csName, csName)) // found overlapping control section name

return true;

cur = cur->next;

}

return false;

}

int searchES(char\* symName) {

NODE\* curCS = extSymTable;

NODE\* curES;

while(curCS) {

curES = ((CNT\_SEC\*)(curCS->data))->extSym;

while(curES) {

if(!strcmp(symName, ((EXT\_SYMBOL\*)(curES->data))->symName))

return ((EXT\_SYMBOL\*)(curES->data))->address; // found external symbol

curES = curES->next;

}

curCS = curCS->next;

}

return -1;

}

## memory.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: memory.h \*

\* File description: Header file for memory related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

short mem[MEM\_SIZE]; // virtual memory (1MB)

void dumpCMD(INPUT\_CMD); // COMMAND: dump

void editCMD(INPUT\_CMD); // COMMAND: edit

void fillCMD(INPUT\_CMD); // COMMAND: fill

void resetCMD(); // COMMAND: reset

## memory.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: memory.c \*

\* File description: Tasks related to virtual memory handing. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "memory.h"

void dumpCMD(INPUT\_CMD ipcmd) {

static int start = 0;

int end, i, j;

if(start >= MEM\_SIZE)

start = 0; // if start address exceeded memory limit, reset to 0x00000

end = start + 159; // set initial end value

if(ipcmd.argCnt) { // if there was argument entered in command

start = hexToDec(ipcmd.arg[0]);

end = start + 159;

if(ipcmd.argCnt == 2) // if there was 2 arguments

end = hexToDec(ipcmd.arg[1]);

}

if(end >= MEM\_SIZE)

end = MEM\_SIZE - 1; // if end address exceed memory limit, set to 0xFFFFF

// start loop from the beginning of each 16 Bytes

// finish loop till end of each 16 Bytes

for(i = start / 16 \* 16; i < (end / 16 + 1) \* 16; i++) {

if(!(i % 16))

printf("%05X ", i); // print address for beginning of each line

if(i < start || i > end)

printf(" "); // do NOT dump memory outside print range

else

printf("%02X ", mem[i]); // dump memory in hexadecimal value

if(!((i + 1) % 16)) { // line finished

printf("; ");

// print content converted into ASCII character

for(j = i - 15; j <= i; j++)

printf("%c", ((j >= start && j <= end) && mem[j] >= 32 && mem[j] <= 126) ? mem[j] : '.');

puts("");

}

}

start = end + 1; // remember last print address

}

void editCMD(INPUT\_CMD ipcmd) {

int add, val;

add = hexToDec(ipcmd.arg[0]); // address to edit

val = hexToDec(ipcmd.arg[1]); // replace value

mem[add] = val;

}

void fillCMD(INPUT\_CMD ipcmd) {

int i, start, end, val;

start = hexToDec(ipcmd.arg[0]); // fill start address

end = hexToDec(ipcmd.arg[1]); // fill end address

val = hexToDec(ipcmd.arg[2]); // fill value

for(i = start; i <= end; i++)

mem[i] = val;

}

void resetCMD() {

int i;

for(i = 0; i < MEM\_SIZE; i++)

mem[i] = 0; // set memory to 0x00000

for(i = 0; i < REG\_CNT; i++)

registers[i] = 0; // initialize registers

progAddr = execAddress = endAddress = 0;

}

## shell.h

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: shell.h \*

\* File description: Header file for shell related tasks. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

void helpCMD(); // COMMAND: help

void dirCMD(); // COMMAND: dir

void quitCMD(); // COMMAND: quit

void histCMD(); // COMMAND: history

void typeCMD(INPUT\_CMD); // COMMAND: type

void histAdd(char\*); // function to add input command into history

## shell.c

/\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*

\* \*

\* Sogang University \*

\* Department of Computer Science and Engineering \*

\* \*

\* Subject name: System Programming \*

\* Project title: [3] SIC/XE Machine - Linking Loader \*

\* \*

\* Author: Inho Kim \*

\* Student ID: 20161577 \*

\* \*

\* File name: shell.c \*

\* File description: Tasks processed in shell environment. \*

\* \*

\* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \* \*/

#include "20161577.h"

#include "linkedList.h"

#include "shell.h"

void helpCMD() {

printf("h[elp]\n"

"d[ir]\n"

"q[uit]\n"

"hi[story]\n"

"du[mp] [start, end]\n"

"e[dit] address, value\n"

"f[ill] start, end, value\n"

"reset\n"

"opcode mnemonic\n"

"opcodelist\n"

"assemble filename\n"

"type filename\n"

"symbol\n"

"progaddr [address]\n"

"loader [object filename1] [object filename2] [...]\n"

"run\n"

"bp [address]\n");

}

void dirCMD() {

DIR\* dir = opendir("."); // current directory

char\* entStr;

char path[258] = "./"; // entry path string

ENTRY\* ent; // entry

STBUF buf; // stat

if(!dir) {

puts("ERROR opening directory...");

return;

}

ent = readdir(dir); // read entry

while(ent) {

path[2] = '\0'; // clear path string

entStr = ent->d\_name; // entry name

stat(strcat(path, entStr), &buf);

printf("%-s", entStr); // print entry name

if(S\_ISDIR(buf.st\_mode)) // check for directory

printf("/");

else if(buf.st\_mode & S\_IXUSR) // check for exec file

printf("\*");

ent = readdir(dir); // read next entry

if(ent)

puts("");

}

closedir(dir);

puts("");

}

void quitCMD() {

puts("Exiting SIC...");

freeList(&histList); // free history linked list

opCodeTableFree(); // free opcode hash table

symTableFree(); // free SYMTAB

parseListFree(); // free ASM parse list

objListFree(); // free object code list

modListFree(); // free modification record list

extSymTableFree(); // free ESTAB

freeList(&breakPntList);// free break point list

exit(0);

}

void histCMD() {

printList(histList, printHistory);

}

void typeCMD(INPUT\_CMD ipcmd) {

FILE\* fp = fopen(ipcmd.arg[0], "r");

char c;

if(!fp) {

puts("ERROR: File not found.");

return;

}

while((c = fgetc(fp)) != EOF)

putchar(c);

if(fclose(fp))

puts("WARNING: Error closing file.");

}

void histAdd(char\* str) {

void\* data = malloc(sizeof(HIST\_NODE));

strcpy(((HIST\_NODE\*)data)->str, str);

addToList(&histList, data);

}

## Makefile

20161577.out: main.o cmdProc.o shell.o memory.o hash.o assembler.o linkLoader.o execute.o linkedList.o

    gcc -Wall -o 20161577.out main.o cmdProc.o shell.o memory.o hash.o assembler.o linkLoader.o execute.o linkedList.o -lm

    @echo "\n>>> To execute, type ./20161577.out\n"

main.o: 20161577.h 20161577.c

    gcc -Wall -c -o main.o 20161577.c -lm

cmdProc.o: 20161577.h cmdProc.c

    gcc -Wall -c -o cmdProc.o cmdProc.c -lm

shell.o: 20161577.h shell.c

    gcc -Wall -c -o shell.o shell.c -lm

memory.o: 20161577.h memory.c

    gcc -Wall -c -o memory.o memory.c -lm

hash.o: 20161577.h hash.c

    gcc -Wall -c -o hash.o hash.c -lm

assembler.o: 20161577.h assembler.c

    gcc -Wall -c -o assembler.o assembler.c -lm

linkLoader.o: 20161577.h linkLoader.h

    gcc -Wall -c -o linkLoader.o linkLoader.c -lm

linkedList.o: 20161577.h linkedList.h shell.h

    gcc -Wall -c -o linkedList.o linkedList.c -lm

execute.o: 20161577.h linkedList.h execute.h

    gcc -Wall -c -o execute.o execute.c -lm

clean:

    -rm \*.o

    -rm 20161577.out